

Exhibit 1

LIBRARY OF CONGRESS

*Copyright Office
of the United States*

WASHINGTON, D.C.

THIS IS TO CERTIFY that the attached photocopies are a true representation of the work entitled **AIROS 5.3** deposited in the Copyright Office with claim of copyright registered under **TXu 1-795-147**.

THIS IS TO CERTIFY FURTHER, that deposits submitted electronically bear no identifying marks.

IN WITNESS WHEREOF, the seal of this Office is affixed hereto on November 2, 2018.

Karyn A. Temple
Acting United States Register of Copyrights and Director



By: Jarletta Walls
Supervisory Copyright Specialist
Records Research and Certification Section
Office of Public Records and Repositories



Use of this material is governed by the U.S. copyright law 17 U.S.C. 101 et seq.

```

        (part_crc_t*)(fwp->data + fwp->data_size);

        crc = htonl(crc32(0L, (unsigned char*)p,
                        fwp->data_size + sizeof(part_t)));
        if (crc != fwp->signature->crc) {
            WARN("Invalid '%s' CRC (claims: %u, but is %u)\n",
fwp->header->name, fwp->signature->crc, crc);
        }
    }

    p = (part_t*)((unsigned char*)p + sizeof(part_t) +
                  ntohl(p->length) + sizeof(part_crc_t));
    /* check bounds */
    if (((unsigned char*)p - base) >= size) {
        return -3;
    }
    ++i;
}
fw->part_count = 1;

sig = (signature_t*)p;
if (strncmp(sig->magic, MAGIC_END, MAGIC_LENGTH) != 0) {
    ERROR("Bad firmware signature\n");
    return -4;
}

crc = htonl(crc32(0L, base, (unsigned char*)sig - base));
if (crc != sig->crc) {
    WARN("Invalid signature CRC (claims: %u, but is %u)\n",
sig->crc, crc);
}

return 0;
}

static int
fw_split(const fw_t* fw, const char* prefix) {
    int i;
    const fw_part_t* fwp;
    FILE* f;
    char filename[PATH_MAX];

    snprintf(filename, sizeof(filename), "%s.txt", prefix);

    INFO("Creating descriptor file:\n%s\n", filename);
    /* write descriptor file */
    f = fopen(filename, "w");
    if (f == NULL) {
        ERROR("Couldn't open file '%s' for writing!\n", filename);
        return -1;
    }

    for (i = 0; i < fw->part_count; ++i) {
        fwp = &fw->parts[i];

        fprintf(f, "%c%c%c%c\t%s\t\t0x%02X\t0x%08X\t0x%08X\t0x%08X\t0x%0
8X\t%s.%s\n",
STRMAGIC(fwp->header->magic), fwp->header->name, ntohl(f
wp->header->part_no),
        ntohl(fwp->header->baseaddr),

```

```

        ntohl(fwp->header->part_len),
        ntohl(fwp->header->memaddr),
        ntohl(fwp->header->entryaddr),
        prefix, fwp->header->name);
    }
    fclose(f);

    INFO("Creating partition data files: \n");

    for (i = 0; i < fw->part_count; ++i) {
        fwp = &fw->parts[i];

        snprintf(filename, sizeof(filename), "%s.%s",
                 prefix, fwp->header->name);
        f = fopen(filename, "w");
        if (f == NULL) {
            ERROR("Failed opening file '%s' for writing: %s\n",
                 filename, strerror(errno));
            continue;
        }

        INFO("\t%s\n", filename);

        if (fwrite(fwp->data, fwp->data_size, 1, f) != 1) {
            ERROR("Failed writing to file '%s': %s\n",
                 filename, strerror(errno));
            fclose(f);
            continue;
        }
        fclose(f);
    }

    return 0;
}

static void
usage(const char* progname) {
    INFO("Usage: %s [options] <firmware file> [<fw file2> ... <fw fileN>]\n"

        "\t-o <output file prefix>\t"
        "\t\t- output file prefix, default: firmware version\n"
        "\t-d\t\t\t- turn debug output on\n"
        "\t-h\t\t\t- this help\n", progname);
}

static int
do_fwsplit(const char* filename) {
    int rc;
    int fd;
    struct stat st;
    fw_t fw;
    unsigned char* addr;

    INFO("Firmware file: '%s'\n", filename);

    rc = stat(filename, &st);
    if (rc) {
        ERROR("Couldn't stat() file '%s': %s\n",

```

```

        usage(argv[0]);
        return -1;
    }

    if (optind >= argc) {
        usage(argv[0]);
        return -1;
    }

    if (strlen(prefix) != 0 && (optind + 1) < argc) {
        WARN("Prefix overridden - will process only the first firmware f
ile\n");
        do_fwsplit(argv[optind]);
    } else {
        for (i = optind; i < argc; ++i) {
            do_fwsplit(argv[i]);
        }
    }

    return 0;
}

----- END fwsplit.c -----
----- BEGIN ubnt-mkfwimage.c -----
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <zlib.h>
#include <sys/mman.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#include "fw.h"

typedef struct part_data {
    char    magic[MAGIC_LENGTH + 1];
    char    partition_name[64];
    int     partition_index;
    size_t  partition_baseaddr;
    size_t  partition_memaddr;
    size_t  partition_entryaddr;
    size_t  partition_length;

    char    filename[PATH_MAX];
    struct stat stats;
} part_data_t;

#define MAX_SECTIONS      8
#define DEFAULT_OUTPUT_FILE    "firmware-image.bin"
#define DEFAULT_VERSION        "UNKNOWN"

#define OPTIONS "hv:o:i:"

// #define DEBUG(...) fprintf(stderr, "DEBUG: " __VA_ARGS__)

```

```

        filename, strerror(errno));
    return -2;
}

if (st.st_size < sizeof(header_t) + sizeof(signature_t)) {
    ERROR("File '%s' is too short\n", filename);
    return -3;
}

fd = open(filename, O_RDONLY);
if (fd < 0) {
    ERROR("Couldn't open file '%s': %s\n",
        filename, strerror(errno));
    return -4;
}

addr = (unsigned char*)mmap(0, st.st_size, PROT_READ, MAP_SHARED, fd, 0);
if (addr == MAP_FAILED) {
    ERROR("Failed mmaping memory for file '%s'\n", filename);
    close(fd);
    return -5;
}

// parse & validate fw
rc = fw_parse(addr, st.st_size, &fw);
if (rc) {
    ERROR("Invalid firmware file '%s'!\n", filename);
    munmap(addr, st.st_size);
    close(fd);
    return -6;
}

if (strlen(prefix) == 0) {
    strncpy(prefix, fw.version, sizeof(prefix));
}
fw_split(&fw, prefix);

munmap(addr, st.st_size);
close(fd);

return 0;
}

int
main(int argc, char* argv[]) {
    int o, i;

    memset(prefix, 0, sizeof(prefix));

    while ((o = getopt(argc, argv, "hdo:")) != -1) {
        switch (o) {
            case 'd':
                debug++;
                break;
            case 'o':
                if (optarg) {
                    strncpy(prefix, optarg, sizeof(prefix));
                }
                break;
            case 'h':

```

```

#define DEBUG(...)
#define INFO(...) fprintf(stdout, __VA_ARGS__)
#define ERROR(...) fprintf(stderr, "ERROR: " __VA_ARGS__)
#define WARN(...) fprintf(stderr, "WARN: " __VA_ARGS__)

typedef struct image_info {
    char version[128];
    char outputfile[PATH_MAX];
    size_t part_count;
    part_data_t parts[MAX_SECTIONS];
} image_info_t;

static void write_header(void* mem, const char* version)
{
    header_t* header = mem;
    memset(header, 0, sizeof(header_t));

    memcpy(header->magic, MAGIC_UBNT_HEADER, MAGIC_LENGTH);
    strncpy(header->version, version, sizeof(header->version));
    header->crc = htonl(crc32(0L, (unsigned char *)header,
                           sizeof(header_t) - 2 * sizeof(u_int32_t)));
    header->pad = 0L;
}

static void write_signature(void* mem, size_t sig_offset)
{
    /* write signature */
    signature_t* sign = (signature_t*)(mem + sig_offset);
    memset(sign, 0, sizeof(signature_t));

    memcpy(sign->magic, MAGIC_END, MAGIC_LENGTH);
    sign->crc = htonl(crc32(0L, (unsigned char *)mem, sig_offset));
    sign->pad = 0L;
}

static int write_part(void* mem, part_data_t* d)
{
    char* addr;
    int fd;
    part_t* p = mem;
    part_crc_t* crc = mem + sizeof(part_t) + d->stats.st_size;

    fd = open(d->filename, O_RDONLY);
    if (fd < 0)
    {
        ERROR("Failed opening file '%s'\n", d->filename);
        return -1;
    }

    if ((addr=(char*)mmap(0, d->stats.st_size, PROT_READ, MAP_SHARED, fd, 0)
) == MAP_FAILED)
    {
        ERROR("Failed mmaping memory for file '%s'\n", d->filename);
        close(fd);
        return -2;
    }

    memcpy(mem + sizeof(part_t), addr, d->stats.st_size);
    munmap(addr, d->stats.st_size);
}

```

```

memcpy(p->magic, d->magic, MAGIC_LENGTH);
memset(p->name, 0, sizeof(p->name));
strcpy(p->name, d->partition_name, sizeof(p->name));
p->length = htonl(d->stats.st_size);
p->part_len = htonl(d->partition_length);
p->part_no = htonl(d->partition_index);
p->baseaddr = htonl(d->partition_baseaddr);
p->entryaddr = htonl(d->partition_entryaddr);
p->memaddr = htonl(d->partition_memaddr);

crc->crc = htonl(crc32(0L, mem, d->stats.st_size + sizeof(part_t)));
crc->pad = 0L;

return 0;
}

static void usage(const char* progame)
{
    INFO("Usage: %s [options]\n"
        "\t-v <version string>\t - firmware version information, default: %s\n"
        "\t-o <output file>\t - firmware output file, default: %s\n"
        "\t-i <input file>\t\t - firmware layout file, default: none\n"
        "\t-h\t\t\t - this help\n",
        progame, DEFAULT_VERSION, DEFAULT_OUTPUT_FILE);
}

static void print_image_info(const image_info_t* im)
{
    int i = 0;
    INFO("Firmware version: '%s'\n"
        "Output file: '%s'\n"
        "Part count: %zu\n",
        im->version, im->outputfile,
        im->part_count);

    for (i = 0; i < im->part_count; ++i)
    {
        const part_data_t* d = &im->parts[i];
        INFO(" [%4s] '%s': %8ld bytes (free: %8ld)\n",
            d->magic,
            d->partition_name,
            d->stats.st_size,
            d->partition_length - d->stats.st_size);
    }
}

/**
 * Image layout file format:
 *
 * <partition name>\t<partition index>\t<partition size>\t<data file name>
 */
static int parse_image_layout(const char* layoutfile, image_info_t* im)
{
    int fd = 0;

```

```

char line[1028];
FILE* f;

im->part_count = 0;

fd = open(layoutfile, O_RDONLY);
if (fd < 0) {
    ERROR("Could not open file '%s'\n", layoutfile);
    return -1;
}

f = fdopen(fd, "r");
if (f == NULL) {
    close(fd);
    return -2;
}

while (!feof(f))
{
    char name[16];
    char magic[MAGIC_LENGTH + 1];
    size_t index;
    size_t baseaddr;
    size_t size;
    size_t memaddr;
    size_t entryaddr;
    char file[PATH_MAX];
    size_t c;
    part_data_t* d;

    if (fgets(line, sizeof(line), f) == NULL)
        break;

    if ((c = sscanf(line, "%4[^\t]\\t%16[^\t]\\t%zX\\t%zX\\t%zX\\t%zX\\t%zX\\t%128[^\t\\n]", magic, name, &index, &baseaddr, &size, &memaddr, &entryaddr, f
ile)) != 8)
        continue;

    DEBUG("%s\\t%s\\t\\t0x%02zX\\t0x%08zX\\t0x%08zX\\t0x%08zX\\t0x%08zX\\t%s\\n", magic, name, index, baseaddr, size, memaddr, entryaddr, file);

    c = im->part_count;
    if (c == MAX_SECTIONS)
        break;

    d = &im->parts[c];
    strncpy(d->magic, magic, sizeof(d->magic));
    strncpy(d->partition_name, name, sizeof(d->partition_name));
    d->partition_index = index;
    d->partition_baseaddr = baseaddr;
    d->partition_length = size;
    d->partition_entryaddr = entryaddr;
    d->partition_memaddr = memaddr;
    strncpy(d->filename, file, sizeof(d->filename));

    im->part_count++;
}

fclose(f);

```

```

    return 0;
}

/**
 * Checks the availability and validity of all image components.
 * Fills in stats member of the part_data structure.
 */
static int validate_image_layout(image_info_t* im)
{
    int i;

    if (im->part_count == 0 || im->part_count > MAX_SECTIONS)
    {
        ERROR("Invalid part count '%zu'\n", im->part_count);
        return -1;
    }

    for (i = 0; i < im->part_count; ++i)
    {
        part_data_t* d = &im->parts[i];
        int len = str_en(d->partition_name);
        if ((len == 0 || len > 16) && !strcmp(d->magic, MAGIC_PART, MAG
IC_LENGTH))
        {
            ERROR("Invalid partition name '%s' of the part %d\n",
                d->partition_name, i);
            return -1;
        }
        if (stat(d->filename, &d->stats) < 0)
        {
            ERROR("Couldn't stat file '%s' from part '%s'\n",
                d->filename, d->partition_name);
            return -2;
        }
        if (d->stats.st_size == 0)
        {
            ERROR("File '%s' from part '%s' is empty!\n",
                d->filename, d->partition_name);
            return -3;
        }
        if ((d->stats.st_size > d->partition_length) && !strcmp(d->magi
c, MAGIC_PART, MAGIC_LENGTH)) {
            ERROR("File '%s' too big (%d) - max size: 0x%08zx (excee
ds %lu bytes)\n",
                d->filename, i, d->partition_length, d->
stats.st_size - d->partition_length);
            return -4;
        }
    }

    return 0;
}

static int build_image(image_info_t* im)
{
    char* mem;
    char* p1r;
    size_t mem_size;
    FILE* f;
    int i;

```

```

// build in-memory buffer
mem_size = sizeof(header_t) + sizeof(signature_t);
for (i = 0; i < im->part_count; ++i)
{
    part_data_t* d = &im->parts[i];
    mem_size += sizeof(part_t) + d->stats.st_size + sizeof(part_crc_t);
};

mem = (char*)calloc(mem_size, 1);
if (mem == NULL)
{
    ERROR("Cannot allocate memory chunk of size '%zu'\n", mem_size);

    return -1;
}

// write header
write_header(mem, im->version);
ptr = mem + sizeof(header_t);
// write all parts
for (i = 0; i < im->part_count; ++i)
{
    part_data_t* d = &im->parts[i];
    int rc;
    if ((rc = write_part(ptr, d)) != 0)
    {
        ERROR("ERROR: failed writing part %u '%s'\n", i, d->part
ition_name);
    }
    ptr += sizeof(part_t) + d->stats.st_size + sizeof(part_crc_t);
}
// write signature
write_signature(mem, mem_size - sizeof(signature_t));

// write in-memory buffer into file
if ((f = fopen(im->outputfile, "w")) == NULL)
{
    ERROR("Can not create output file: '%s'\n", im->outputfile);
    return -10;
}

if (fwrite(mem, mem_size, 1, f) != 1)
{
    ERROR("Could not write %zu bytes into file: '%s'\n",
        mem_size, im->outputfile);
    return -11;
}

free(mem);
fclose(f);
return 0;
}

int main(int argc, char* argv[])
{
    char inputfile[PATH_MAX];
    int o, rc;

```

```

image_info_t im;

memset(&im, 0, sizeof(im));
memset(inputfile, 0, sizeof(inputfile));

strcpy(im.outputfile, DEFAULT_OUTPUT_FILE);
strcpy(im.version, DEFAULT_VERSION);

while ((o = getopt(argc, argv, OPTIONS)) != -1)
{
    switch (o) {
        case 'v':
            if (optarg)
                strncpy(im.version, optarg, sizeof(im.version));

            break;
        case 'o':
            if (optarg)
                strncpy(im.outputfile, optarg, sizeof(im.outputf
file));

            break;
        case 'i':
            if (optarg)
                strncpy(inputfile, optarg, sizeof(inputfile));

            break;
        case 'h':
            usage(argv[0]);
            return -1;
    }
}

if (strlen(inputfile) == 0)
{
    ERROR("Input file is not specified, cannot continue\n");
    usage(argv[0]);
    return -2;
}

if ((rc = parse_image_layout(inputfile, &im)) != 0)
{
    ERROR("Failed parsing firmware layout file '%s' - error code: %d
\n",
        inputfile, rc);
    return -3;
}

if ((rc = validate_image_layout(&im)) != 0)
{
    ERROR("Failed validating firmware layout - error code: %d\n", rc
);
    return -4;
}

print_image_info(&im);

if ((rc = build_image(&im)) != 0)
{
    ERROR("Failed building image file '%s' - error code: %d\n", im.o
utputfile, rc);
    return -5;
}

```

```
    }  
    return 0;  
}  
----- END ubnt-mkfwimage.c -----
```

```

----- BEGIN AboutDialog.java -----
/*
 * Copyright (c) 2008-2012 UBiQUITi Networks, Inc.
 *
 * All Rights Reserved. Unpublished rights reserved under the copyright laws
 * of the United States. The software contained on this media is proprietary
 * to and embodies the confidential technology of UBiQUITi Networks, Inc. The
 * possession or receipt of this information does not convey any right to
 * disclose its contents, reproduce it, use it, or license its use, for
 * manufacture or sale. The foregoing restriction applies to the information or
 *
 * anything described therein. Any use, disclosure, or reproduction without
 * UBiQUITi's prior written permission is strictly prohibited.
 *
 */
package com.ubnt.app;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.net.URL;

import javax.swing.GroupLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

import com.ubnt.util.StringUtils;
import com.ubnt.util.ui.UiUtils;

/**
 * The class displays the about dialog box with version information, etc.
 *
 * @author Ramin
 */
public class AboutDialog extends JDialog
{
    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = 1L;

    /** The logo image loc. */
    public static String logoImageLoc = "/images/about.png";

    // Variables declaration
    /** The buttons panel. */
    private JPanel buttonsPanel;

    /** The content panel. */

```

```

private JPanel contentPanel;

/** The edition label. */
private JLabel editionLabel;

/** The background panel. */
private JPanel backgroundPanel;

/** The ok button. */
private JButton okButton;

/** The url label. */
private JLabel urlLabel;

/** The version label. */
private JLabel versionLabel;

/** The build info label. */
private JLabel buildInfoLabel;

/** The app url. */
private String appUrl;

/** The company logo. */
private Image companyLogo;

/**
 * Creates new form AboutDialog.
 *
 * @param parent the parent
 * @param modal the modal
 */
public AboutDialog (java.awt.Frame parent, boolean modal)
{
    super (parent, modal);
    URL url = AboutDialog.class.getResource (logoImageLoc);
    ImageIcon temp = new ImageIcon (url);
    if(temp != null) {
        companyLogo = temp.getImage ();
    }
    initComponents ();
}

/**
 * The Class ImagePanel.
 */
public class ImagePanel extends JPanel {

    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = 1L;

    /** The img. */
    private Image img;

    /**
     * Instantiates a new image panel.
     *
     * @param img the img
     */
    ImagePanel (Image img) {

```

```

        this.img = img;
        setOpaque(false);
    }

    /* (non-Javadoc)
     * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
     */
    public void paintComponent (Graphics g) {
        g.drawImage(img, 0, 0, this);
        super.paintComponent(g);
    }

    /* (non-Javadoc)
     * @see javax.swing.JComponent#getPreferredSize()
     */
    public Dimension getPreferredSize () {
        return new Dimension (img.getWidth (this), img.getHeight (th
is));
    }

    /**
     * This method is called from within the constructor to initialize the f
orm.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    private void initComponents ()
    {
        backgroundPanel = new ImagePanel (companyLogo);
        contentPanel = new JPanel ();
        versionLabel = new JLabel ();
        editionLabel = new JLabel ();
        urlLabel = new JLabel ();
        urlLabel.setCursor (Cursor.getPredefinedCursor (Cursor.HAND_CURS
OR));

        buttonsPanel = new JPanel ();
        okButton = new JButton ();
        buildInfoLabel = new JLabel ();
        appUrl = System.getProperty ("app.url");
        if(appUrl == null) {
            appUrl = "http://www.ubnt.com";
        }
        setDefaultCloseOperation (WindowConstants.DISPOSE_ON_CLOSE);
        String appName = System.getProperty ("app.name");
        if(appName == null) {
            appName = "AirView";
        }
        setTitle ("About " + appName); // NOI18N
        setAlwaysOnTop (true);
        setIconImage (null);

        Font plainFont = new Font ("Verdana", java.awt.Font.PLAIN, 10);
        Font boldFont = new Font ("Verdana", java.awt.Font.BOLD, 10);

        backgroundPanel.setName("backgroundPanel");
        backgroundPanel.setLayout (new java.awt.BorderLayout ());
        getContentPane ().setLayout (new java.awt.BorderLayout ());
    }

```

```

E_START);

        getContentPane ().add(backgroundPanel, java.awt.BorderLayout.PAG

contentPanel.setName ("contentPanel"); // NOI18N
contentPanel.setOpaque(false);
versionLabel.setFont (boldFont); // NOI18N
versionLabel.setHorizontalAlignment (SwingConstants.LEFT);
versionLabel.setText ("Version: " + System.getProperty ("app.ver
sion") +
        (!StringUtils.isEmpty (System.getProperty ("app.minorVer
sion")) ? "." + System.getProperty ("app.minorVersion") : "") +
        (!StringUtils.isEmpty (System.getProperty ("app.release"
)) ? " " + System.getProperty ("app.release") : ""));
versionLabel.setName ("versionLabel"); // NOI18N
versionLabel.setOpaque(false);
versionLabel.setForeground(Color.WHITE);

        editionLabel.setFont (boldFont); // NOI18N
        editionLabel.setHorizontalAlignment (SwingConstants.LEFT);
        if(AirViewer.isViewer()) {
            editionLabel.setText ("Embedded Edition");
        } else {
            editionLabel.setText ("AirView Manager");
        }
        editionLabel.setName ("editionLabel"); // NOI18N
        editionLabel.setOpaque(false);
        editionLabel.setForeground(Color.WHITE);

        urlLabel.setFont (plainFont); // NOI18N
        urlLabel.setHorizontalAlignment (SwingConstants.LEFT);
        urlLabel.setText ("<html><a color=white href='" + appUrl + "'>"
+ appUrl + "</a></html>");
        urlLabel.setName ("urlLabel"); // NOI18N
        urlLabel.setOpaque(false);
        urlLabel.setForeground(Color.WHITE);
        urlLabel.addMouseListener (new MouseAdapter () {
            public void mouseClicked (final MouseEvent evt)
            {
                SwingUtilities.invokeLater (new Runnable () {pub
lic void run () {
                    urlLabelMouseClicked (evt);
                });
            }
        });
        buttonsPanel.setName ("buttonsPanel"); // NOI18N
        buttonsPanel.setLayout (new java.awt.FlowLayout (java.awt.FlowLa
yout.CENTER));
        buttonsPanel.setOpaque(false);

        okButton.setText ("Close");
        okButton.setName ("okButton"); // NOI18N
        okButton.addActionListener (new java.awt.event.ActionListener ()
        {
            public void actionPerformed (java.awt.event.ActionEvent
            {
                okButtonActionPerformed (evt);
            }
        });
        buttonsPanel.add (okButton);

```

```

        buildInfoLabel.setFont (plainFont); // NOI18N
        buildInfoLabel.setText("Build Info: " + Application.BUILD_INFO);

        buildInfoLabel.setName("buildInfoLabel"); // NOI18N
        buildInfoLabel.setOpaque(false);
        buildInfoLabel.setForeground(Color.WHITE);

        GroupLayout contentPanelLayout = new GroupLayout (contentPanel);

        contentPanel.setLayout (contentPanelLayout);
        contentPanelLayout.setHorizontalGroup (contentPanelLayout.create
ParallelGroup (
            GroupLayout.Alignment.LEADING).addGroup (
            contentPanelLayout.createSequentialGroup ().addGap (85,
85, 85).addGroup (
                contentPanelLayout.createParallelGroup (GroupLayo
out.Alignment.CENTER)
                    .addComponent (versionLabel, GroupLayout
.DEFAULT_SIZE, 182, Short.MAX_VALUE)
                    .addComponent (editionLabel, GroupLayout
.DEFAULT_SIZE, 182, Short.MAX_VALUE)
                    .addComponent (buildInfoLabel, GroupLayo
ut.DEFAULT_SIZE, 182, Short.MAX_VALUE)
                    .addComponent (urlLabel, GroupLayout.DEF
AULT_SIZE, 182, Short.MAX_VALUE)
                ).addComponent (buttonsPanel, GroupLayo
ut.DEFAULT_SIZE, 182, Short.MAX_VALUE));
        contentPanelLayout.setVerticalGroup (contentPanelLayout.createPa
rallelGroup (
            GroupLayout.Alignment.LEADING).addGroup (
            contentPanelLayout.createSequentialGroup ()
                .addGap(150, 150, 150)
                .addComponent(versionLabel)
                .addGap(2, 2, 2)
                .addComponent(editionLabel)
                .addGap(2, 2, 2)
                .addComponent(buildInfoLabel)
                .addGap(25, 25, 25)
                .addComponent(urlLabel)
                .addGap(13, 13, Short.MAX_VALUE)
                .addComponent(buttonsPanel)
                .addGap(6, 6, 6)
            );

        backgroundPanel.add (contentPanel, java.awt.BorderLayout.CENTER)
;

        setSize(companyLogo.getWidth (this)+2, companyLogo.getHeight (th
is) - 2);

        setResizable(false);
        pack ();

        UiUtils.centerComponent (this);
    } // </editor-fold> // GEN-END: initComponents

/**
 * Ok button action performed.
 *
 * @param evt the evt

```

```

    */
    private void okButtonActionPerformed (java.awt.event.ActionEvent evt)
    {
        // GEN-FIRST:event_okButtonActionPerformed
        this.setVisible (false);
        this.dispose ();
        // GEN-LAST:event_okButtonActionPerformed

    /**
     * Url label mouse clicked.
     *
     * @param evt the evt
     */
    private void urlLabelMouseClicked (java.awt.event.MouseEvent evt)
    {
        // GEN-FIRST:event_urlLabelMouseClicked
        // GEN-LAST:event_urlLabelMouseClicked

    /**
     * The main method.
     *
     * @param args the command line arguments
     */
    public static void main (String args[])
    {
        java.awt.EventQueue.invokeLater (new Runnable () {
            public void run ()
            {
                AboutDialog dialog = new AboutDialog (new JFrame
                ()), true);

                dialog.addWindowListener (new
                java.awt.event.WindowAdapter () {
                    public void windowClosing (java.awt.event.
                    WindowEvent e)
                    {
                        System.exit (0);
                    }
                });
                dialog.setVisible (true);
            }
        });
    }

}
----- END AboutDialog.java -----
----- BEGIN AirViewer.java -----
/*
 * Copyright (c) 2008-2012 UBIQUITI Networks, Inc.
 *
 * All Rights Reserved. Unpublished rights reserved under the copyright laws
 * of the United States. The software contained on this media is proprietary
 * to and embodies the confidential technology of UBIQUITI Networks, Inc. The
 * possession or receipt of this information does not convey any right to
 * disclose its contents, reproduce it, use it, or license its use, for
 * manufacture or sale. The foregoing restriction applies to the information or
 * anything described therein. Any use, disclosure, or reproduction without
 * UBIQUITI's prior written permission is strictly prohibited.
 */
package com.ubnt.app;

```

```

/*
import com.centerkey.utils.BareBonesBrowserLaunch;
*/
import com.ubnt.chart.data.xy.UbntXIntervalSeries;
import com.ubnt.chart.renderer.xy.UbntXYAreaRenderer;
import java.text.DecimalFormat;
/*
import com.ubnt.device.ChannelDefinition;
import com.ubnt.device.Product;
*/
import com.ubnt.device.RssiFrameImpl;
import com.ubnt.device.BasicSpan;
import com.ubnt.device.Channel;
import com.ubnt.device.ChainMask;
import com.ubnt.device.RFScanDevice;
import com.ubnt.device.RFScanDeviceInfo;
import com.ubnt.device.RFScanDeviceLocator;
import com.ubnt.device.RFScanRange;
import com.ubnt.device.RssiFrame;
import com.ubnt.device.RssiFrameConsumer;
import com.ubnt.device.remote.*;
import com.ubnt.device.remote.discovery.ConnectDialog;
import com.ubnt.device.remote.discovery.LangStrings;
import com.ubnt.device.remote.discovery.net.LiteStationQueryServer;
import com.ubnt.device.remote.discovery.net.QueryServer;
import com.ubnt.device.remote.discovery.net.Scanner;

import com.ubnt.util.FileUtils;

/*
import com.ubnt.util.HelpLauncher;
*/
import com.ubnt.util.HzMath;
import com.ubnt.util.SpringFramework;
import com.ubnt.util.ui.BorderlessStatusWindow;
import com.ubnt.util.ui.GraphicsPanel;
import com.ubnt.util.ui.PaintDelegate;
import com.ubnt.util.ui.UiUtils;

/*
import org.apache.commons.lang.ArrayUtils;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.math.stat.StatUtils;
*/
import org.apache.log4j.Logger;

import org.jfree.chart.ChartMouseEvent;
import org.jfree.chart.ChartMouseListener;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.ChartRenderingInfo;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.annotations.XYAnnotation;
import org.jfree.chart.annotations.XYShapeAnnotation;
import org.jfree.chart.annotations.XYTextAnnotation;
import org.jfree.chart.axis.AxisLocation;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.block.BlockBorder;

```

```

import org.jfree.chart.block.BlockContainer;
import org.jfree.chart.block.BlockFrame;
import org.jfree.chart.block.BorderArrangement;
import org.jfree.chart.block.EmptyBlock;
import org.jfree.chart.block.FlowArrangement;
import org.jfree.chart.block.LabelBlock;
import org.jfree.chart.entity.ChartEntity;
import org.jfree.chart.labels.ItemLabelAnchor;
import org.jfree.chart.labels.ItemLabelPosition;
import org.jfree.chart.labels.StandardXYItemLabelGenerator;
import org.jfree.chart.labels.XYToolTipGenerator;
import org.jfree.chart.plot.CombinedDomainXYPlot;
import org.jfree.chart.plot.DatasetRenderingOrder;
import org.jfree.chart.plot.IntervalMarker;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.SeriesRenderingOrder;
import org.jfree.chart.plot.ValueMarker;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.LookupPaintScale;
import org.jfree.chart.renderer.xy.StandardXYItemRenderer;
import org.jfree.chart.renderer.xy.XYAreaRenderer;
import org.jfree.chart.renderer.xy.XYBarRenderer;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYShapeRenderer;
import org.jfree.chart.title.CompositeTitle;
import org.jfree.chart.title.LegendTitle;
import org.jfree.chart.title.PaintScaleLegend;
import org.jfree.chart.title.TextTitle;

import org.jfree.data.xy.DefaultXYZDataset;
import org.jfree.data.xy.XIntervalSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.data.xy.YWithXInterval;

import org.jfree.ui.HorizontalAlignment;
import org.jfree.ui.Layer;
import org.jfree.ui.RectangleEdge;
import org.jfree.ui.RectangleInsets;
import org.jfree.ui.TextAnchor;
import org.jfree.ui.VerticalAlignment;

import org.springframework.context.ApplicationEvent;
import org.springframework.context.ApplicationListener;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.ItemSelectable;
import java.awt.Paint;
import java.awt.Point;

```

```

import java.awt.Shape;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.InputEvent;
import java.awt.event.ItemEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import java.math.BigDecimal;

import java.net.URISyntaxException;
import java.net.URL;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
import java.util.SortedMap;
import java.util.Timer;
import java.util.TimerTask;
import java.util.TreeMap;
import javax.swing.JFileChooser;
import javax.swing.filechooser.*;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.MenuElement;
import javax.swing.JPopupMenu;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JToggleButton;
import javax.swing.JToolBar;

```

```

import javax.swing.KeyStroke;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;

import java.util.logging.LogManager;

/**
 * The Class AirViewer.
 */
public class AirViewer extends JFrame implements RssiFrameConsumer, ApplicationL
istener {

    /** The LOGGER. */
    private static Logger LOGGER;

    /** The chart title font. */
    public static Font chartTitleFont = new Font("Dialog", Font.BOLD, 11);

    /** The Constant PREFS_COMMENTS. */
    private static final String PREFS_COMMENTS = "\n" +
        " AirView saved preferences. Please be very careful with any manual\n" +
        " edits as wrong/unexpected values can cause application instability.\n\
n" +
        " NOTE: If you want to disable/comment-out a particular value, simply\n\
" +
        " add a .BAK at the end of its attribute name and the application will\n\
" +
        " effectively ignore it. Putting a hash (#) in front of the attribute n
ame\n" +
        " will not work as the application will delete that entry upon next save
.\n\n";

    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = 1L;

    /** The viewer. */
    private static AirViewer viewer;

    /** The j content pane. */
    private JPanel jContentPane = null;

    /** The graphs panel. */
    private JPanel graphsPanel = null;

    /** The channel usage panel. */
    private ChartPanel channelUsagePanel = null;

    /** The chart panel2. */
    private ChartPanel chartPanel2 = null;

    /** The realtime chart panel. */
    private ChartPanel realtimeChartPanel = null;

    /** The config menu item. */
    private JMenuItem configMenuItem;

```

Exhibit 2

2 * Copyright (C) 2007 Ubiquiti Networks, Inc.

3 *

4 * This program is free software; you can redistribute it and/or
5 * modify it under the terms of the GNU General Public License as
6 * published by the Free Software Foundation; either version 2 of the
7 * License, or (at your option) any later version.

8 *

9 * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.

13 *

14 * You should have received a copy of the GNU General Public License
15 * along with this program; if not, write to the Free Software
16 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

17 */

18
19 #include <sys/types.h>
20 #include <sys/stat.h>

21 #include <fcntl.h>

22 #include <unistd.h>

23 #include <string.h>

24 #include <errno.h>

25 #include <zlib.h>

26 #include <sys/mman.h>

27 #include <netinet/in.h>

28 #include <stdio.h>

29 #include <stdlib.h>

30 #include <limits.h>

31

32 #include "fw.h"

33

34 typedef struct part_data {
35 char partition_name[64];
36 int partition_index;
37 u_int32_t partition_baseaddr;
38 u_int32_t partition_memaddr;
39 u_int32_t partition_entryaddr;
40 u_int32_t partition_length;
4142 char filename[PATH_MAX];
43 struct stat stats;

44 } part_data_t;

45

46 #define MAX_SECTIONS 8

47 #define DEFAULT_OUTPUT_FILE "firmware-image.bin"

48 #define DEFAULT_VERSION "UNKNOWN"

49

50 #define OPTIONS "hv:o:i:"

51

52 static int debug = 0;

53

54 typedef struct image_info {
55 char version[256];
56 char outputfile[PATH_MAX];
57 u_int32_t part_count;
58 part_data_t parts[MAX_SECTIONS];
59 } image_info_t;

60

61 static void write_header(void* mem, const char* version)

62 {

63 header_t* header = mem;
64 memset(header, 0, sizeof(header_t));

65

66 memcpy(header->magic, MAGIC_HEADER, MAGIC_LENGTH);
67 strncpy(header->version, version, sizeof(header->version));68 header->crc = htonl(crc32(0L, (unsigned char *)header,
69 sizeof(header_t) - 2 * sizeof(u_int32_t)));

```

70     case 1: pad = 0; break;
71 }
72
73
74 static void write_signature(void* mem, u_int32_t sig_offset)
75 {
76     /* write signature */
77     signature_t* sign = (signature_t*)(mem + sig_offset);
78     memset(sign, 0, sizeof(signature_t));
79
80     memcpy(sign->magic, MAGIC_END, MAGIC_LENGTH);
81     sign->crc = htonl(crc32(0L, (unsigned char *)mem, sig_offset));
82     sign->pad = 0L;
83 }
84
85 static int write_part(void* mem, part_data_t* d)
86 {
87     char* addr;
88     int fd;
89     part_t* p = mem;
90     part_crc_t* crc = mem + sizeof(part_t) + d->stats.st_size;
91
92     fd = open(d->filename, O_RDONLY);
93     if (fd < 0)
94     {
95         ERROR("Failed opening file '%s'\n", d->filename);
96         return -1;
97     }
98
99     if ((addr=(char*)mmap(0, d->stats.st_size, PROT_READ, MAP_SHARED, fd, 0)) ==
100     MAP_FAILED)
101     {
102         ERROR("Failed mmaping memory for file '%s'\n", d->filename);
103         close(fd);
104         return -2;
105     }
106
107     memcpy(mem + sizeof(part_t), addr, d->stats.st_size);
108     munmap(addr, d->stats.st_size);
109
110     memset(p->name, 0, sizeof(p->name));
111     strncpy(p->magic, MAGIC_PART, MAGIC_LENGTH);
112     strncpy(p->name, d->partition_name, sizeof(p->name));
113     p->index = htonl(d->partition_index);
114     p->data_size = htonl(d->stats.st_size);
115     p->part_size = htonl(d->partition_length);
116     p->baseaddr = htonl(d->partition_baseaddr);
117     p->memaddr = htonl(d->partition_memaddr);
118     p->entryaddr = htonl(d->partition_entryaddr);
119
120     crc->crc = htonl(crc32(0L, mem, d->stats.st_size + sizeof(part_t)));
121     crc->pad = 0L;
122
123     return 0;
124 }
125
126 static void usage(const char* progname)
127 {
128     INFO("Version %s\n"
129         "Usage: %s [options]\n"
130         "\t-v <version string>\t - firmware version information, default: %s\n"
131         "\t-o <output file>\t - firmware output file, default: %s\n"
132         "\t-i <input file>\t\t - firmware layout file, default: none\n"
133         "\t-h\t\t\t - this help\n", VERSION,
134         progname, DEFAULT_VERSION, DEFAULT_OUTPUT_FILE);
135 }
136
137 static void print_image_info(const image_info_t* im)
138 {

```

```

138     INFO("Firmware version: '%s'\n"
139         "Output file: '%s'\n"
140         "Part count: %u\n",
141         im->version, im->outputfile,
142         im->part_count);
143
144     for (i = 0; i < im->part_count; ++i)
145     {
146         const part_data_t* d = &im->parts[i];
147         INFO(" %10s: %8ld bytes (free: %8ld)\n",
148             d->partition_name,
149             d->stats.st_size,
150             d->partition_length - d->stats.st_size);
151     }
152 }
153
154
155
156
157 /**
158  * Image layout file format:
159  *
160  * <partition name>\t<partition index>\t<partition size>\t<data file name>
161  *
162  */
163 static int parse_image_layout(const char* layoutfile, image_info_t* im)
164 {
165     int fd = 0;
166     char line[1024];
167     FILE* f;
168
169     im->part_count = 0;
170
171     fd = open(layoutfile, O_RDONLY);
172     if (fd < 0) {
173         ERROR("Could not open file '%s'\n", layoutfile);
174         return -1;
175     }
176
177     f = fdopen(fd, "r");
178     if (f == NULL) {
179         close(fd);
180         return -2;
181     }
182
183     while (!feof(f))
184     {
185         char name[32];
186         u_int32_t index;
187         u_int32_t baseaddr;
188         u_int32_t size;
189         u_int32_t memaddr;
190         u_int32_t entryaddr;
191         char file[PATH_MAX];
192         u_int32_t c;
193         part_data_t* d;
194
195         if (fgets(line, sizeof(line), f) == NULL)
196             break;
197
198         // TODO: very inconvenient format, use smarter parsing someday
199         if ((c = sscanf(line, "%32[^\t]\t%X\t%X\t%X\t%X\t%X\t128[^\t\n]", name,
200             &index, &baseaddr, &size, &memaddr, &entryaddr, file)) != 7)
201             continue;
202
203         DEBUG("%s\t\t0x%02X\t0x%08X\t0x%08X\t0x%08X\t0x%08X\t%s\n", name, index,
204             baseaddr, size, memaddr, entryaddr, file);
205
206         c = im->part_count;

```

```

205         break;
206     }
207
208     d = &im->parts[c];
209     strncpy(d->partition_name, name, sizeof(d->partition_name));
210     d->partition_index = index;
211     d->partition_baseaddr = baseaddr;
212     d->partition_length = size;
213     d->partition_memaddr = memaddr;
214     d->partition_entryaddr = entryaddr;
215     strncpy(d->filename, file, sizeof(d->filename));
216
217     im->part_count++;
218 }
219
220 fclose(f);
221
222 return 0;
223 }
224
225 /**
226  * Checks the availability and validity of all image components.
227  * Fills in stats member of the part_data structure.
228  */
229 static int validate_image_layout(image_info_t* im)
230 {
231     int i;
232
233     if (im->part_count == 0 || im->part_count > MAX_SECTIONS)
234     {
235         ERROR("Invalid part count '%d'\n", im->part_count);
236         return -1;
237     }
238
239     for (i = 0; i < im->part_count; ++i)
240     {
241         part_data_t* d = &im->parts[i];
242         int len = strlen(d->partition_name);
243         if (len == 0 || len > 16)
244         {
245             ERROR("Invalid partition name '%s' of the part %d\n",
246                 d->partition_name, i);
247             return -1;
248         }
249         if (stat(d->filename, &d->stats) < 0)
250         {
251             ERROR("Couldn't stat file '%s' from part '%s'\n",
252                 d->filename, d->partition_name);
253             return -2;
254         }
255         if (d->stats.st_size == 0)
256         {
257             ERROR("File '%s' from part '%s' is empty!\n",
258                 d->filename, d->partition_name);
259             return -3;
260         }
261         if (d->stats.st_size > d->partition_length) {
262             ERROR("File '%s' too big (%d) - max size: 0x%08X (exceeds %lu bytes)\n",
263                 d->filename, i, d->partition_length,
264                 d->stats.st_size - d->partition_length);
265             return -4;
266         }
267     }
268
269     return 0;
270 }
271
272 static int build_image(image_info_t* im)
273 {

```

```

274     char* mem;
275     char* ptr;
276     u_int32_t mem_size;
277     FILE* f;
278     int i;
279
280     // build in-memory buffer
281     mem_size = sizeof(header_t) + sizeof(signature_t);
282     for (i = 0; i < im->part_count; ++i)
283     {
284         part_data_t* d = &im->parts[i];
285         mem_size += sizeof(part_t) + d->stats.st_size + sizeof(part_crc_t);
286     }
287
288     mem = (char*)calloc(mem_size, 1);
289     if (mem == NULL)
290     {
291         ERROR("Cannot allocate memory chunk of size '%u'\n", mem_size);
292         return -1;
293     }
294
295     // write header
296     write_header(mem, im->version);
297     ptr = mem + sizeof(header_t);
298     // write all parts
299     for (i = 0; i < im->part_count; ++i)
300     {
301         part_data_t* d = &im->parts[i];
302         int rc;
303         if ((rc = write_part(ptr, d)) != 0)
304         {
305             ERROR("ERROR: failed writing part %u '%s'\n", i, d->partition_name);
306         }
307         ptr += sizeof(part_t) + d->stats.st_size + sizeof(part_crc_t);
308     }
309     // write signature
310     write_signature(mem, mem_size - sizeof(signature_t));
311
312     // write in-memory buffer into file
313     if ((f = fopen(im->outputfile, "w")) == NULL)
314     {
315         ERROR("Can not create output file: '%s'\n", im->outputfile);
316         return -10;
317     }
318
319     if (fwrite(mem, mem_size, 1, f) != 1)
320     {
321         ERROR("Could not write %d bytes into file: '%s'\n",
322             mem_size, im->outputfile);
323         return -11;
324     }
325
326     free(mem);
327     fclose(f);
328     return 0;
329 }
330
331
332 int main(int argc, char* argv[])
333 {
334     char inputfile[PATH_MAX];
335     int o, rc;
336     image_info_t im;
337
338     memset(&im, 0, sizeof(im));
339     memset(inputfile, 0, sizeof(inputfile));
340
341     strcpy(im.outputfile, DEFAULT_OUTPUT_FILE);
342     strcpy(im.version, DEFAULT_VERSION);

```

```
343 while ((o = getopt(argc, argv, OPTIONS)) != -1)
344 {
345     switch (o) {
346         case 'v':
347             if (optarg)
348                 strncpy(im.version, optarg, sizeof(im.version));
349             break;
350         case 'o':
351             if (optarg)
352                 strncpy(im.outputfile, optarg, sizeof(im.outputfile));
353             break;
354         case 'i':
355             if (optarg)
356                 strncpy(inputfile, optarg, sizeof(inputfile));
357             break;
358         case 'h':
359             usage(argv[0]);
360             return -1;
361     }
362 }
363
364 if (strlen(inputfile) == 0)
365 {
366     ERROR("Input file is not specified, cannot continue\n");
367     usage(argv[0]);
368     return -2;
369 }
370
371 if ((rc = parse_image_layout(inputfile, &im)) != 0)
372 {
373     ERROR("Failed parsing firmware layout file '%s' - error code: %d\n",
374         inputfile, rc);
375     return -3;
376 }
377
378 if ((rc = validate_image_layout(&im)) != 0)
379 {
380     ERROR("Failed validating firmware layout - error code: %d\n", rc);
381     return -4;
382 }
383
384 print_image_info(&im);
385
386 if ((rc = build_image(&im)) != 0)
387 {
388     ERROR("Failed building image file '%s' - error code: %d\n", im.outputfile, rc);
389     return -5;
390 }
391
392 return 0;
393 }
394
395
```

Exhibit 3

2 * Copyright (C) 2007 Ubiquiti Networks, Inc.

3 *

4 * This program is free software; you can redistribute it and/or
5 * modify it under the terms of the GNU General Public License as
6 * published by the Free Software Foundation; either version 2 of the
7 * License, or (at your option) any later version.

8 *

9 * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.

13 *

14 * You should have received a copy of the GNU General Public License
15 * along with this program; if not, write to the Free Software
16 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

17 */

18
19 #define _GNU_SOURCE
20 #include <string.h>
21 #include <stdio.h>
22 #include <netinet/in.h>
23 #include <sys/types.h>
24 #include <fcntl.h>
25 #include <errno.h>
26 #include <sys/stat.h>
27 #include <sys/mman.h>
28 #include <zlib.h>
29 #include <limits.h>
30 #include <unistd.h>

31

32
33 #include "fw.h"

34

35 static int debug = 0;
36 static char prefix[PATH_MAX];

37

38 #define MAX_PARTS 8

39

40 typedef struct fw_part {
41 part_t* header;
42 unsigned char* data;
43 u_int32_t data_size;
44 part_crc_t* signature;
45 } fw_part_t;

46

47 typedef struct fw {
48 u_int32_t size;
49 char version[256];
50 fw_part_t parts[MAX_PARTS];
51 int part_count;
52 } fw_t;

53

54 static int
55 fw_check_header(const header_t* h) {
56 u_int32_t crc;
57 int len = sizeof(header_t) - 2 * sizeof(u_int32_t);
58
59 crc = crc32(0L, (unsigned char*)h, len);
60 DEBUG("Calculated CRC: 0x%08X, expected: 0x%08X\n", htonl(crc), h->crc);
61 if (htonl(crc) != h->crc)
62 return -1;
63
64 return 0;
65 }

66

67 static int
68 fw_parse(const unsigned char* base, unsigned long size, fw_t* fw) {
69 const header_t* h = (const header_t*)base;

```

70     base = (part_t*)(base + sizeof(header_t));
71     signature_t* sig;
72     u_int32_t crc;
73     int i = 0;
74
75     if (fw == NULL)
76         return -1;
77
78     if (fw_check_header(h)) {
79         return -2;
80     }
81     memset(fw, 0, sizeof(fw_t));
82     fw->size = size;
83     memcpy(fw->version, h->version, sizeof(fw->version));
84     INFO("Firmware version: '%s'\n", fw->version);
85
86     p = (part_t*)(base + sizeof(header_t));
87     i = 0;
88     while (strcmp(p->magic, MAGIC_END, MAGIC_LENGTH) != 0) {
89         DEBUG("Partition (%c%c%c%c): %s [%u]\n",
90             p->magic[0], p->magic[1], p->magic[2], p->magic[3],
91             p->name, ntohl(p->index));
92         DEBUG(" Partition size: 0x%X\n", ntohl(p->part_size));
93         DEBUG("  Data size: %u\n", ntohl(p->data_size));
94
95         if ((strcmp(p->magic, MAGIC_PART, MAGIC_LENGTH) == 0) && (i < MAX_PARTS)) {
96             fw_part_t* fwp = &fw->parts[i];
97
98             fwp->header = p;
99             fwp->data = (unsigned char*)p + sizeof(part_t);
100             fwp->data_size = ntohl(p->data_size);
101             fwp->signature =
102                 (part_crc_t*)(fwp->data + fwp->data_size);
103
104             crc = htonl(crc32(0L, (unsigned char*)p,
105                 fwp->data_size + sizeof(part_t)));
106             if (crc != fwp->signature->crc) {
107                 WARN("Invalid '%s' CRC (claims: %u, but is %u)\n",
108                     fwp->header->name, fwp->signature->crc, crc);
109             }
110         }
111
112         p = (part_t*)((unsigned char*)p + sizeof(part_t) +
113             ntohl(p->data_size) + sizeof(part_crc_t));
114
115         /* check bounds */
116         if (((unsigned char*)p - base) >= size) {
117             return -3;
118         }
119         ++i;
120     }
121     fw->part_count = i;
122
123     sig = (signature_t*)p;
124     if (strcmp(sig->magic, MAGIC_END, MAGIC_LENGTH) != 0) {
125         ERROR("Bad firmware signature\n");
126         return -4;
127     }
128
129     crc = htonl(crc32(0L, base, (unsigned char*)sig - base));
130     if (crc != sig->crc) {
131         WARN("Invalid signature CRC (claims: %u, but is %u)\n",
132             sig->crc, crc);
133     }
134
135     return 0;
136 }
137
138 static int

```

```

139 fw_part_t* fw = &fw_parts[0];
140 int i;
141 const fw_part_t* fwp;
142 FILE* f;
143 char filename[PATH_MAX];
144
145 snprintf(filename, sizeof(filename), "%s.txt", prefix);
146
147 INFO("Creating descriptor file:\n\t%s\n", filename);
148 /* write descriptor file */
149 f = fopen(filename, "w");
150 if (f == NULL) {
151     ERROR("Couldn't open file '%s' for writing!\n", filename);
152     return -1;
153 }
154
155 for (i = 0; i < fw->part_count; ++i) {
156     fwp = &fw->parts[i];
157
158     fprintf(f, "%s\t\t0x%02X\t0x%08X\t0x%08X\t0x%08X\t0x%08X\t%s.\n",
159             fwp->header->name,
160             ntohl(fwp->header->index),
161             ntohl(fwp->header->baseaddr),
162             ntohl(fwp->header->part_size),
163             ntohl(fwp->header->memaddr),
164             ntohl(fwp->header->entryaddr),
165             prefix, fwp->header->name);
166
167 }
168 fclose(f);
169
170 INFO("Creating partition data files: \n");
171
172 for (i = 0; i < fw->part_count; ++i) {
173     fwp = &fw->parts[i];
174
175     snprintf(filename, sizeof(filename), "%s.%s",
176              prefix, fwp->header->name);
177     f = fopen(filename, "w");
178     if (f == NULL) {
179         ERROR("Failed opening file '%s' for writing: %s\n",
180              filename, strerror(errno));
181         continue;
182     }
183
184     INFO("\t%s\n", filename);
185
186     if (fwrite(fwp->data, fwp->data_size, 1, f) != 1) {
187         ERROR("Failed writing to file '%s': %s\n",
188              filename, strerror(errno));
189         fclose(f);
190         continue;
191     }
192     fclose(f);
193 }
194
195 return 0;
196 }
197
198 static void
199 usage(const char* progname) {
200     INFO("Version %s\n"
201          "Usage: %s [options] <firmware file> [<fw file2> ... <fw fileN>]\n"
202          "\t-o <output file prefix>\n"
203          "    - output file prefix, default: firmware version\n"
204          "\t-d\t\t - turn debug output on\n"
205          "\t-h\t\t - this help\n", VERSION, progname);
206 }

```

```

208
209 static int
210 do_fwsplit(const char* filename) {
211     int rc;
212     int fd;
213     struct stat st;
214     fw_t fw;
215     unsigned char* addr;
216
217     INFO("Firmware file: '%s'\n", filename);
218
219     rc = stat(filename, &st);
220     if (rc) {
221         ERROR("Couldn't stat() file '%s': %s\n",
222             filename, strerror(errno));
223         return -2;
224     }
225
226     if (st.st_size < sizeof(header_t) + sizeof(signature_t)) {
227         ERROR("File '%s' is too short\n", filename);
228         return -3;
229     }
230
231     fd = open(filename, O_RDONLY);
232     if (fd < 0) {
233         ERROR("Couldn't open file '%s': %s\n",
234             filename, strerror(errno));
235         return -4;
236     }
237
238     addr=(unsigned char*)mmap(0, st.st_size, PROT_READ, MAP_SHARED, fd, 0);
239     if (addr == MAP_FAILED) {
240         ERROR("Failed mmaping memory for file '%s'\n", filename);
241         close(fd);
242         return -5;
243     }
244
245     // parse & validate fw
246     rc = fw_parse(addr, st.st_size, &fw);
247     if (rc) {
248         ERROR("Invalid firmware file '%s'!\n", filename);
249         munmap(addr, st.st_size);
250         close(fd);
251         return -6;
252     }
253
254     if (strlen(prefix) == 0) {
255         strncpy(prefix, fw.version, sizeof(prefix));
256     }
257     fw_split(&fw, prefix);
258
259     munmap(addr, st.st_size);
260     close(fd);
261
262     return 0;
263 }
264
265 int
266 main(int argc, char* argv[]) {
267     int o, i;
268
269     memset(prefix, 0, sizeof(prefix));
270
271     while ((o = getopt(argc, argv, "hdo:")) != -1) {
272         switch (o) {
273             case 'd':
274                 debug++;
275                 break;
276             case 'o':

```

```
277         strncpy(prefix, optarg, sizeof(prefix));
278     }
279     break;
280 case 'h':
281     usage(argv[0]);
282     return -1;
283 }
284 }
285 }
286
287 if (optind >= argc) {
288     usage(argv[0]);
289     return -1;
290 }
291
292 if (strlen(prefix) != 0 && (optind + 1) < argc) {
293     WARN("Prefix overridden - will process only the first firmware file\n");
294     do_fwsplit(argv[optind]);
295 } else {
296     for (i = optind; i < argc; ++i) {
297         do_fwsplit(argv[i]);
298     }
299 }
300
301 return 0;
302 }
303
```